HHS Public Access

Author manuscript

Mol Reprod Dev. Author manuscript; available in PMC 2017 May 12.

Published in final edited form as:

Mol Reprod Dev. 2015 ; 82(7-8): 518–529. doi:10.1002/mrd.22489.

The ImageJ ecosystem: an open platform for biomedical image analysis

Johannes Schindelin, **Curtis T. Rueden**, **Mark C. Hiner**, and **Kevin W. Eliceiri***

Laboratory for Optical and Computational Instrumentation Room 271 Animal Sciences 1675 Observatory Drive University of Wisconsin at Madison Madison, WI 53706 USA

Abstract

Technology in microscopy advances rapidly, enabling increasingly affordable, faster, and more precise quantitative biomedical imaging, which necessitates correspondingly more-advanced image processing and analysis techniques. A wide range of software is available – from commercial to academic, special-purpose to Swiss army knife, small to large–but a key characteristic of software that is suitable for scientific inquiry is its accessibility. Open-source software is ideal for scientific endeavors because it can be freely inspected, modified, and redistributed; in particular, the open-software platform ImageJ has had a huge impact on life sciences, and continues to do so. From its inception, ImageJ has grown significantly due largely to being freely available and its vibrant and helpful user community. Scientists as diverse as interested hobbyists, technical assistants, students, scientific staff, and advanced biology researchers use ImageJ on a daily basis, and exchange knowledge via its dedicated mailing list. Uses of ImageJ range from data visualization and teaching to advanced image processing and statistical analysis. The software's extensibility continues to attract biologists at all career stages as well as computer scientists who wish to effectively implement specific image-processing algorithms. In this review, we use the ImageJ project as a case study of how open-source software fosters its suites of software tools, making multitudes of image-analysis technology easily accessible to the scientific community. We specifically explore what makes ImageJ so popular, how it impacts life science, how it inspires other projects, and how it is self-influenced by coevolving projects within the ImageJ ecosystem.

Keywords

microscopy; software; open source; fiji; image processing

Introduction

Ever since digital-imaging equipment entered the world of science, life scientists have collaborated with computer scientists to apply image-processing techniques to analyze biomedical data. The aim is to use computational processes to accelerate repetitive tasks

*Corresponding author: Kevin W. Eliceiri, Phone: (608) 263-8481, eliceiri@wisc.edu.

Quote: “ImageJ's extensible structure means new functionality can always be layered on top of the existing tools rather than being created from the ground up.”

while also obtaining quantitative results, since statistical results are much more compelling, scientifically speaking, than qualitative observations. As the field of image processing matured (e.g., Castleman 1996), computer-vision experts developed specialized techniques that could be applied to biomedical images.

Biomedical image processing is a subset of computer-vision research with its own specific challenges – namely, low-light conditions required to keep the imaged specimen alive. Compared to conveniently uniform footage (e.g., from a video camera) biomedical images require substantial knowledge about the physical intricacies of the optics involved, coupled with textbook computer vision expertise, for sound image processing. Advances in biomedical image-processing techniques have allowed for the corroboration of research outcomes by quantifying them in a rigid, statistical manner while simultaneously raising the bar for life science research when it comes to substantiating scientific observations. Thus, life scientists needed accessible methods to execute image-processing and analysis techniques to quantitatively support their research.

A large number of reviews have been written on the subject of biomedical image processing and analysis tools in general (e.g., Eliceiri et al. 2012) and on tools to perform specific tasks (e.g., Meijering et al. 2006; Pham et al. 2000). There are also entire books focusing on the topic of image processing written for specific scientific disciplines; in microscopy alone, readers can find excellent resources for video image processing (Inoue 1981), digital microscopy (Sluder and Wolf 2007), and digital imaging in optical microscopy (http://micro.magnet.fsu.edu/primer/digitalimaging/index.html). Rather than paraphrasing these excellent resources, this review will focus on the ImageJ project (Schneider et al. 2012) as a case study of how open-source software fosters an ecosystem of software tools, making an abundance of image-analysis methods and approaches easily accessible to the scientific community.

ImageJ facilitates scientific inquiry

In 1987, Wayne Rasband, who at that time was working at the National Institute of Health (Bethesda, MD, USA), released “NIH Image”, the predecessor to ImageJ. NIH Image entered a field already crowded with highly advanced scientific image-processing software that was targeted at computer scientists (e.g., Cristy et al. 1994; Konstantinides and Rasure 1994). A few characteristics set this program apart from the competitors, though, and these nuances are maintained today: application and source code were available free of cost, it had a very simple user interface, and could run on affordable desktop machines. Additionally, as a project that welcomes source-code contributions, ImageJ attracts end users and software programmers alike.

The open and inclusive software architecture was very much by design, as Wayne Rasband's understanding was and remains that the NIH, being funded through public money, must serve the public good. As a consequence, ImageJ remains open source, making it as scientifically accessible and relevant as possible: the program's functionality can be changed based on user needs, its inner workings can be scrutinized, and its architecture provides an excellent resource for learning how to implement image-processing algorithms. If instead ImageJ was offered for free, but its source code was made unavailable or restrictively

licensed, the program would essentially become a “black box” that hinders interested parties from fully understanding and validating what it does – a concept that is simply incompatible with the very nature of scientific research.

The accessibility of ImageJ makes it an ideal teaching resource. The unlimited number of free downloads of this program obviate the otherwise prohibitively expensive per-seat licenses necessary to establish and maintain classroom computers; indeed, course participants can even use their own laptops, minimizing the effort to apply what has been learned in their daily work. In addition, macros and plugins dedicated to demonstrating image-analysis concepts are easily added to ImageJ. One such example is the Spirals macro (http://imagej.net/Spirals), which is designed to dispel the all-too-common notion that it is reliable to quantify colors by eye. The educational aspects of ImageJ are not limited to teaching the fundamentals of image processing, though; it is one of the most popular tools for teaching the development of image-processing algorithms (Burger and Burge 2010) because its open-source architecture is readily extended using plugins written in Java (requiring only a Java Development Kit that is also available free of cost). Indeed, the Biomedical Imaging Group of the École Polytechnique Fédérale de Lausanne (e.g., Forster et al. 2004; Delgado-Gonzalo et al. 2012) offers a regular semester-long image-processing course using ImageJ to teach students how to implement image-processing algorithms, which leverages both strengths of this software.

The ImageJ community also set an example for how research can be more effective, by establishing an interdisciplinary forum where knowledge is disseminated and questions that might appear difficult to experts in one scientific field may be easily answered by an expert in another field. The diversity of the expertise present on ImageJ's mailing list (http://imagej.net/Mailing_Lists), ranging from experimental biologists to paleontologists to astronomers to computer scientists, give rise to daily, insightful scientific exchanges conducted in a polite tone. Such a collaborative spirit may have had the biggest impact on how life-science research is performed today, as it enables and inspires the way other scientific projects are run. More and more scientists acknowledge the fact that software plays a role as important in research as the materials and methods. Just as protocols or genetic mutant lines are expected to be shared with other scientists to facilitate validation and subsequent research, source code used to produce scientific findings must be made available to other scientists. Given only the text of a scientific publication, it would be too cumbersome for an independent group to reproduce the findings within it without easy access to the same materials and methods; indeed, even if these requirements were met, there would still be no guarantee that the experiments could ever be reproduced faithfully. This same philosophy applies to software: given only a description of an algorithm, it is not only an undue burden to ask independent groups who seek to validate the published results to re-implement the algorithm themselves, there would be no relation whatsoever between the original and the independent implementation, rendering any attempt at a validating the original work invalid.

ImageJ is the heart of a software ecosystem

The parallels between the technological landscape of an open-source software project and a biological ecosystem are numerous. The freedom to modify and redistribute open-source software leads to a proliferation of different software “species”, each occupying its own specialized niche. Projects diverge (or *fork*, in programming terminology) over time, giving rise to mutants. Components can interoperate with and benefit from one another in a continuous, co-operational evolution or are found instead to compete with each other, providing multiple solutions to similar problems.

Continuous advances in scientific understanding necessitate a constant state of flux in scientific software: new projects crop up, old projects are succeeded and retired, whereas collaborative projects proceed to answer scientific questions and ensure their mutual longevity. This process is facilitated by a vibrant community of scientists exchanging ideas, knowledge, advice, and programs on public channels, such as the ImageJ mailing list. By offering the program for free, encouraging participation, and setting an example through a gentle and constructive presence, Wayne Rasband undoubtedly helped the ImageJ community to flourish and to develop a culture that truly follows the scientific tradition that let Newton state: “If I have seen further, it is by standing on the shoulders of giants.” Every scientific discovery is based on other scientific research, and knowledge sharing provides fertile soil for new discoveries.

From a technical perspective, ImageJ provides a user interface with functions to load, display, and save images; basic image-processing functionality, such as convolution filters; and an extension mechanism including support for plugins and macros. Its extensibility is truly the root of ImageJ's effectiveness: advanced image-processing methods (e.g., wavelet analysis or active contour segmentation) are not provided by the core application, but rather made available as third-party plugins by specialists in the corresponding fields. ImageJ provides a so-called Application Programming Interface (API) for these plugins, which are special-purpose software components that extend ImageJ's functionality by offering additional commands via menu entries. Plugins can tap into ImageJ's existing functionality (e.g., reading and writing images or requesting user input via dialogs, displaying histograms, plots, and spreadsheets) without having to implement the same functionality from scratch, a process often labeled as “reinventing the wheel” by software developers. ImageJ also has the ability to record and replay macros, which are lists of simple instructions that describe the actions that are performed by a user and which ImageJ can interpret to repeat exactly, thereby dramatically increasing the versatility of the program. Macros enable scientists to automate their image processing workflows, to provide their colleagues with exact documentation of their methods, and to collaborate on improving analysis – all with little to no formal training.

ImageJ is not only an application, it is also what software developers call a software library: its functionality can be used without displaying ImageJ's user interface by calling functions directly via the API from other Java programs. This architecture makes it possible for scientists to develop special-purpose, simplified user interfaces that react to user input by calling for ImageJ to execute the actual image processing behind the scenes. The API also makes it possible to perform heavy-duty computing jobs in the background, or to send them

off to a cluster of computers or even a cloud service. Such interoperability between ImageJ and other software packages enables remarkable cross-pollination.

The ImageJ ecosystem is diverse

Thanks to the characteristics described above, ImageJ has created a very large ecosystem in which the most basic organisms are the macros and plugins (see http://imagej.net/plugins/). These plugins are highly varied, capable of modifying existing functionality (e.g., support for a new file format) or introducing completely new operations (e.g., an unsupervised classification algorithm). Such diversity of tools is one of ImageJ's key strengths: plugins can be freely combined to accomplish complex analyses; users can select between multiple plugins offering similar functionality, choosing the most appropriate tool for each situation; and, in contrast to a monolithic one-software approach, ImageJ's extensible structure means new functionality can always be layered on top of the existing tools rather than being created from the ground up.

Considering the expanse of software projects that function within ImageJ, we will focus on the features that distinguish one from one another. The most obvious distinction between plugins is simply their functionality: within image processing, the most common plugin categories are visualization, preprocessing, segmentation, registration, and tracking.

Visualization—Given the ubiquity of images as data within the life sciences, it may be surprising that data visualization is subtly complex. We often dangerously over-estimate our ability to visually quantitate and differentiate (for example, http://imagej.net/Adelsons_Squares). Nonetheless, qualitative visualization is a vital technique that is harnessed through plugins such as 3D Viewer (Schmid et al. 2010) and Volume Viewer (http://imagej.net/Volume_Viewer) that provide hardware-accelerated, interactive three-dimensional rendering within ImageJ. Using these plugins, scientists can assemble three-dimensional structures from a collection of images, allowing the inspection of volumetric regions in the context of their surroundings. Although these plugins may not, themselves, be quantitating pixel values for scientific benefit, the additional perspectives provided by these models can be invaluable in identifying regions of interest.

Preprocessing—Algorithms that alter image pixels (e.g., enhancing contrast, reducing noise, or subtracting background signal) are often required to facilitate subsequent analysis, but care is needed to avoid introducing artifacts by inappropriately preprocessing images. For example, a general Contrast-Limited Adaptive Histogram Equalization (CLAHE) plugin (see http://imagej.net/CLAHE) might be used to make the borders of cellular structures more distinct (see Fig 1), which then makes segmentation possible. Various types of image artifacts and noise often end up with their own optimized plugin(s); in time-based image analysis, for example, photobleaching can be a significant problem, but the Bleach Correction (http://imagej.net/Bleach_Correction) plugin provides a possible correction for this effect.

Segmentation—The automated or assisted identification of regions or structures of interest within an image allows for quantification and further analysis of the desired

structures. The Trainable Weka Segmentation plugin (Kaynig et al. 2010) allows users to define classes of objects within an image. These classes are then applied to the original image, creating a fully segmented dataset (see Fig 2). Specialized segmentation plugins such as the Simple Neurite Tracer (Longair et al. 2011), itself built on the 3D Viewer plugin, can provide a more automated experience by leveraging pre-trained models (e.g. identifying tube-like structures in three-dimensional stacks, in the case of the neurite tracer).

Registration—. Ensuring that images of different samples or different views of the same sample into a common coordinate system (i.e. overlaying them for easier comparison) is the goal of image registration. This seemingly simple action is an important operation in biological image analysis, particularly when a montage or tiled grid of images representing a single, complete sample needs to be analyzed. The Stitching plugin (Preibisch et al. 2009) is a popular ImageJ option for combining such image collections into a single, cohesive output (see Fig 3). The Register Virtual Stack Slices plugin (http://fiji.sc/Register_Virtual_Stack_Slices) provides similar functionality, with the added option to transform the composite image around a base reference image, and offers a variety of registration techniques, including elastic implementation (Arganda-Carreras 2008).

Tracking—Following structures of interest, which were typically identified via segmentation, over a time series is how object tracking is defined. While some of these techniques rely heavily on manual interaction, others can be made automatic and robust enough for high-throughput analysis. The Tool for Automated Sporozoite Tracking (ToAST) plugin (Hegge et al. 2009) is unique in that it was derived from another tracking plugin, MTrack2 (http://imagej.net/MTrack2), specifically for malaria research, but was generalized for binary images. TrackMate (http://imagej.net/TrackMate) is one of the most extensive plugins for ImageJ, providing a robust tracking interface for users, and a consumable API for developers to further extend TrackMate.

Image acquisition—Before images can be analyzed using the techniques above, they must be acquired and digitized. While ImageJ has proven to be a valuable tool for scientific fields as diverse as paleontology or astronomy, it is especially popular in biomedical-image processing. Indeed, NIH Image and ImageJ were originally developed for images obtained using light microscopes – as users are reminded of based on the ImageJ icon, a beautiful Hartnack microscope. It is not surprising, then, that already very early in ImageJ's history, scientists wished to use ImageJ not only to process and analyze images but also to acquire them. Starting with special-purpose plugins supporting certain Hamamatsu cameras or the Scion frame grabbers, scientists developed support to control cameras from within ImageJ and to acquire images. Yet, with each new plugin came a new and different way to control microscope hardware. The μManager project (Stuurman et al. 2007) was started to unify the control interfaces, soon becoming a boon for many microscopy facilities whose staff were tired of training and retraining scientists on how to use an ever-increasing number of user interfaces developed by competing hardware vendors. The open-source structure of μManager allows for frequent contributions from scientific and hardware vendor developers alike, which add support for more and more hardware such as shutters, stages, and cameras. It also proves invaluable as the starting point for new imaging technologies (e.g., the

OpenSPIM platform (Pitrone et al. 2013) which makes selective-plane illumination microscopy (SPIM) accessible for everyday use in life science and to develop the technology further).

Development paradigms—Although functionality is an obvious criterion for differentiating plugins, it is also important to consider the various motivations for plugin development and how each contributes to the greater ecosystem. Many plugins (such as Stitching, mentioned above) were developed for a specific publication and are offered in the interest of facilitating science. These typically add new functionality to ImageJ, and are freely distributed as a contribution to the scientific community. Depending on how these plugins are funded, however, they may or may not continue to be supported over time with fixes and optimizations. In contrast, other plugins are actively developed and maintained, often supporting an ongoing scientific project. The goal of tool maintenance is typically broader, leading to the creation of suites of tools that are capable of solving a wide variety of tasks. Such plugin suites (e.g., those of LOCI, MOSAIC Group, and BioVoxxel) require a significant commitment of resources to develop and maintain: often one or more full-time developer, backed by a university or commercial research group.

Underlying all of these plugin paradigms is the concept of openness. As ImageJ itself is provided free for the benefit of the scientific community, many projects open their source code and welcome community input and feedback. Yet there is also a wide range of plugins that are available free of cost, but without source code; these are often accompanied by a license that restricts or prohibits modification and redistribution. Such plugins are often met with wariness in the community since closed code resides in a “black box” with unknown functionality, which thus cannot truly be reproduced. That said, it appears that more and more formerly closed projects are recognizing this flaw, and are being opened up to collaborate and interoperate with other scientific groups and software for the greater benefit of scientific research. For example, the Biomedical Imaging Group Differentials plugin and ImageScience plugin codes both became openly available in September, 2010 thanks to the ImageJ community.

Although ImageJ itself, a product of the United States government, resides in the public domain free of copyright limitations, there is also space in the ImageJ ecosystem for commercial interests. For example, bridges to commercial software allowing ImageJ plugins in Imaris (http://bitplane.com/imaris/imaris) and MATLAB (http://mathworks.com/products/matlab/) are very much in line with the collaborative spirit of the ImageJ community. Indeed, independent consultants and companies now write special-purpose ImageJ plugins for hire and/or offer training in general-purpose image processing based on ImageJ or for highly-specialized, advanced image analysis (e.g., using the Trainable Segmentation). There are also hybrid groups mixing scientific and commercial interests (e.g. OMERO, an image database system (Allan et al. 2012) and KNIME, a data analytics platform (Berthold et al. 2008)) which offer their software as open-source but also have a commercial arm offering paid consulting. In total, the ImageJ community benefits from these sources of users and developers; indeed, there are public contributions intended to facilitate contract work (e.g., http://git.io/egvhIw), as well as instances of private work that is subsequently distributed as open source plugins, such as Bob Dougherty's OptiNav (http://optinav.com/imagej.html).

While the plugins described above positively interact with each other –such as by running preprocessing macros to prepare an image for registration– most of them are not explicitly tied to one another. Rather, there is a wealth of available plugins from which to choose, allowing users to select what is most appropriate for a particular task.

Co-operational evolution of software components

At first glance, the relationship between ImageJ and its plugins may appear commensal: the ImageJ platform benefits its plugins, but not necessarily vice versa. The mutual evolution of the application, the plugins, and the community, however, is what makes ImageJ a true ecosystem and not just an extendable platform. Several related phenomena exemplify this sort of mutuality: The first is when the needs of a plugin influence the development directions of the core platform. For example, the ObjectJ project (http://simon.bio.uva.nl/objectj/) offers facilities to annotate, segment, track, and analyze time-series images. ObjectJ was developed using ImageJ's then-burgeoning macro language, which at the time was not powerful enough to do everything ObjectJ needed. Consequently, ImageJ's macro capabilities matured into a more powerful programming tool partly due to feature requests made by ObjectJ's developer.

Another mutuality scenario is when a useful feature originally developed for a plugin proves generally useful enough to migrate “upstream” into the core application itself, a fate that is extremely common within the ImageJ ecosystem. The TurboReg plugin (http://bigwww.epfl.ch/thevenaz/turboreg/), for example, defined the concept of point selections, which proved so useful they eventually moved to the ImageJ core application to the benefit of all future plugins. Another example is the Image5D plugin (http://imagej.net/Image5D) for visualization of five-dimensional images (*x, y, z*, time, and channels) which became so popular that it inspired ImageJ to introduce analogous “hyperstack” and “composite image” functionality into its core. A third case is the Fiji project (Schindelin et al. 2012), which implemented many components that have since migrated into core ImageJ, including the Command Finder, Script Editor, ImageJ Updater, ImageJ Launcher, and many others.

A third mutuality is when some functionality is so broadly useful that it makes sense to package it as its own software library, independent from ImageJ. By using this option, a library's capabilities can be harnessed by ImageJ plugins, but also by software from other ecosystems. Several major software libraries have emerged this way:

Bio-Formats—The Bio-Formats project (Linkert et al. 2010) was started out of the need to open images saved in proprietary image file formats, such as the *.lsm* or *.oib* formats written, respectively, by Zeiss’ or Olympus’ software. The question of how to read and sometimes write data stored in such formats is not a scientific one, yet is essential to scientific research: data acquired by microscopes needs to be accessible in order to be processed and analyzed. Bio-Formats supports a wide range of proprietary formats, as well as standard file formats not generally associated with life science, such as movie files; indeed, the infrastructure developed to support a microscope vendors’ proprietary file format is flexible enough to easily add support for non-scientific file formats, too. Bio-Formats can be used as an image input/output library by other plugins, but it is not strictly required since the Bio-Formats

Importer (see http://imagej.net/Bio-Formats) can open any supported data into a general ImageJ structure, which can then be provided as input to any analysis routine.

ImgLib2—The ImgLib2 library (Pietzsch et al. 2012) came into development when it became clear that the scientific community was in need of a powerful *n*-dimensional data processing library that could allow computer-vision experts to implement their algorithms in a very generic way. Historically, ImageJ supported only a handful of data types (unsigned 8-bit, unsigned 16-bit, floating point 32-bit, and 24-bit integer-packed red-green-blue [RGB]), was limited to at most five predefined dimensions (*x, y, z*, time, and channel), and required the complete pixel data to live in computer memory. As a consequence: most plugins supporting more than one data type had to implement the same algorithm multiple times (once per supported data type); were unable to handle new image modalities, such as spectral lifetime, multiple angles, etcetera; and they simply could not handle images larger than the machine's memory could accommodate. All of these issues have been addressed by ImgLib2, making it possible to write generic algorithms that will run, unmodified –even on data types yet to be invented– on an arbitrary number of dimensions and size. Images can be processed without the need to be loaded completely into memory, and can even be stored in remote databases or generated on the fly, without having to modify or recompile the source code of the processing algorithm.

TrakEM2—One of the largest ImageJ plugin libraries, TrakEM2 (Cardona et al. 2012), was designed to work on very large mosaics of electron-microscopy images that would be assembled automatically and displayed in a manner similar to Google maps: pre-calculated zoom levels allow fluid navigation of datasets comprising tens of gigabytes of data. Intended for following neurons through large electron-microscopy recordings, TrakEM2 provides a user interface for interactive segmentation and annotation, with an API for further extension by the community. At the same time, given its scope and utility, TrakEM2 underscored the need for a new answer to the question: How can this plugin get into the hands of the desired users and developers?

Dispersal of software components

With the exception of code distributed as part of ImageJ itself, the classic model for plugin distribution was to require users to individually download the plugin (often from a website hosted by the plugin's developer or institution) and install it manually to a local directory discoverable by ImageJ. Advertising a new plugin to users was limited to the ImageJ mailing list or word of mouth. This scenario was fairly fragile as websites could be taken down, or messages buried in archives; manual plugin installation is just another opportunity for error, especially for plugins with third-party dependencies. Considering these factors, it is not surprising that distributions of carefully curated plugin collections arose.

One of the first plugin collections was known as MBF ImageJ. Assembled by Tony Collins at Babraham, later maintained at the Wright Cell Imaging Facility, and then at McMaster's Biophotonics Facility, this collection provided a large number of hand-selected plugins that proved useful for working with light-microscopy images. Perhaps more significantly, MBF

ImageJ also included a coherent manual, making the collection extremely popular – until its website became defunct in 2012.

MBF ImageJ inspired the Fiji distribution of ImageJ for the life sciences, including a community-driven documentation wiki at http://fiji.sc/, which was subsequently expanded to become a central ImageJ documentation resource at http://imagej.net/. The Fiji project also includes the Fiji Cookbook (http://fiji.sc/Cookbook), a collection of image analysis "recipes" closely modeled after the original MBF ImageJ manual for microscopy. Fiji was originally conceived to address the need of neuroscientists, then later including cell biologists, to distribute a number of ImageJ plugins highly relevant to their research that are developed actively, but independently from each other. This required the design and implementation of a general-purpose method to keep all plugins conveniently up-to-date. For technical reasons, it also required the implementation of a general-purpose ImageJ launcher, which led to the development of several other non-neuroscience-specific features such as a Command Finder and a plugin for submitting bug reports.

Over time, projects such as the 3D Viewer, the Simple Neurite Tracer, and bUnwarpJ (Arganda-Carreras 2008) came under the Fiji-distribution umbrella. To accelerate the development of new software components, an extensible scripting framework was designed within Fiji, allowing power users to write and execute scripts written in Python, Ruby, JavaScript, and other programming languages. Today, Fiji's primary role is to be a "batteries included" distribution of ImageJ – a flagship example of plugin distribution (see http://fiji.sc/Fiji%27s_Menu) and maintenance within the ImageJ community. The Fiji project continues to evolve within the ecosystem, offering a clear set of requirements (http://fiji.sc/Fiji_contribution_requirements) for including plugins within Fiji distribution as well as an active group of maintainers who assist the community in doing so.

Mutations of ImageJ

Changing an existing program and/or developing it independently from the original project (*forking*, or mutating in evolutionary terms) is sometimes frowned upon because the action divides forces, and sometimes improvements are not shared between forked projects. Yet it is important to realize that every time software is modified, every time a local development version differs from the official (or latest-released) version, it is technically a forked project. Thus the challenge is to reconcile the changes at appropriate times, particularly after completing a new feature or a bug fix. Again, the biological analogy is apt: projects of the same "species" have relatively minor differences and can share improvements or merge back together later (i.e., "interbreed" to produce an offspring with the strengths of both) whereas projects with major differences may ultimately diverge into different incompatible and competing species.

Just as with speciation, software-project forks occupy a spectrum of kinship. On one end are projects like ImageJA (http://imagej.net/ImageJA), a very "shallow" fork of ImageJ that is kept closely synchronized with the original, and exists only to provide a unified revision history and build system. Years ago, ImageJA diverged further from ImageJ, providing several technical improvements such as improved support for applets, but most of those

changes have since migrated into ImageJ itself, resulting in ImageJA and ImageJ essentially becoming the same animal again.

On the other end of the spectrum are “diverging” forks such as Bio7, an integrated development environment for ecological modeling, scientific image analysis, and statistical analysis (Austenfeld and Beyschlag 2012), and SalsaJ (http://euhou.net/index.php/salsaj-software-mainmenu-9), software that facilitates teaching astronomy in the classroom. Diverging from ImageJ allowed these programs to develop a distinct user experience: Bio7 enhances ImageJ's programming capabilities by embedding the ImageJ user interface within a rich client platform while SalsaJ hides image processing methods not suited for astronomy images to make typical operations more accessible. The downside is that bug fixes and overall improvements to ImageJ's code base do not trickle down to such forks automatically, and require a substantial maintenance effort by the fork's developers. Even when the software is open source, as is the case with Bio7, there is significant work required to transfer or “port” changes between the two projects, which can elicit increasingly infrequent updates and eventual “speciation” of the project.

ImageJ2: a macromutation

Over the last fifteen years, ImageJ has grown organically as requested features have been added, with many contributions from outside developers. The result has been a program with a wide range of functionality capable of solving a diverse collection of image-processing and analysis problems, particularly in the life sciences. Yet this pattern of growth, even when carefully managed, is no substitute for a holistically engineered package built from the ground up with a modular design. Indeed, any successful software project, after a period of sustained growth and the addition of functionality outside the scope of the program's original intent, benefits from an extended period of examination and refactoring; ImageJ is no exception.

In 2009, NIH funded ImageJ2 (http://imagej.net/), a project to create a new version of ImageJ that would be better able to handle the next generation of multi-dimensional image data. ImageJ2 provides a unified way to call ImageJ commands from other (even non-Java) software applications, supporting unlimited data types and sizes driven by the ImgLib2 library, as well as extensible data input/output driven by the SCIFIO library (http://scif.io/), a generalization of Bio-Formats beyond the life sciences. In evolutionary terms, ImageJ2 can be seen as a macromutation or adaptation of ImageJ: a large jump with many internal changes and improvements that are intended to better serve the needs of the community. In particular, ImageJ2 provides an extensible data engine intended to accommodate new imaging paradigms, such as combined spectral-lifetime imaging (SLIM), selective-plane illumination (SPIM), polarized light microscopy, and large-tiled image mosaics. The project is backed by a robust software design intended to “future-proof” ImageJ as technology continues to advance. The challenge has been to “harden” the ImageJ application without alienating or fragmenting its widespread user base, who relies on it as an everyday research tool. As such, much of the development of ImageJ2 has stressed not only innovation but also compatibility with legacy code, placing a strong emphasis on supporting existing macros and plugins, thereby avoiding the disruption of established uses of ImageJ as a scientific tool

while at the same time offering a wholly redesigned set of software libraries to meet the demands and challenges of current and future research.

ImageJ2 builds on ImageJ's primary strength –extensibility– by providing a richer and more diverse set of plugin interfaces. It also offers a robust update-site mechanism (see http://imagej.net/Update_Sites), which provides an effective venue for developers to publicize their works, from which users can pick and choose new features to install. The Fiji distribution of ImageJ (see “Dispersal of software components” above) is structured as one such update site, and is continually updated to accommodate the gradual migration to ImageJ2. In total, since the introduction of the update-sites feature two years ago, over 100 public update sites have been created (http://sites.imagej.net/, http://imagej.net/List_of_update_sites).

ImageJ2 also addresses the concern of continuity: ImageJ was originally maintained and developed by a single person, who is now enjoying retirement. Exploring new software development and project management techniques –such as distributed source code control, standardized dependency management, automated regression testing, continuous integration, and public issue tracking systems– ImageJ now meets the demands of scalable and distributed software development, facilitating its continued development and maintenance.

SciJava: mutualistic symbiosis

As is typical in the realm of computer software, there are many applications with similar, overlapping, or complementary functionality to that of ImageJ, but designed with distinct and diverging requirements. These applications exist at the periphery of the ImageJ ecosystem, often centered within ecosystems of their own. They range from very general-purpose tool kits – such as R (http://r-project.org/), VisAD (Hibbard 1998), and MATLAB– to more domain-specific packages –such as Icy (de Chaumont et al. 2012), Vaa3D (Long et al. 2012), and Endrov (Henriksson et al. 2013) in the field of biomedial-image informatics. Many of these tools have a focus different than image processing, but still contribute to the ImageJ ecosystem by providing and consuming functionality from each other. As such, users often combine multiple such programs to achieve results not possible with a single software package alone (see Fig 4). This approach is particularly effective when the respective software tools are built with interoperability in mind: for example, some projects –including CellProfiler (Carpenter et al. 2006), MiToBo (Möller et al. 2011), KNIME, and OMERO– already support execution of ImageJ commands within their own paradigms, a trend that we foresee application developer increasingly doing in the future. Even some purely commercial software packages –such as Imaris and MATLAB, via MIJ (Sage et al. 2012)– have facilities for harnessing ImageJ and its plugins for analysis.

One effective tactic for fostering interoperability is the “hackathon”: an extended brainstorm session bringing together programmers of related scientific projects, with the goal of uninterrupted, high-energy exchange of ideas and source code. In a sense, a hackathon is a highly condensed form of how ImageJ's community operates. During one such hackathon in 2011, the SciJava project (http://scijava.org/) was born: a pledge across several different software projects to work together, share common code, and foster interoperability with one another. One major outcome of this pledge has been that non-image-specific functionality,

which was originally developed as part of ImageJ2 and Fiji, has now been generalized into components of SciJava. Of particular note, ImageJ2's application container, improved plugin mechanism, and parameterized module framework became the basis for the SciJava Common library while Fiji's support for various scripting languages became the SciJava scripting framework. Both ImageJ2 and SCIFIO are now built on the SciJava Common library, meaning they share the same foundation, improvements to which benefit both projects. And some ImageJ plugins have begun to take advantage of the SciJava plugin framework as well: for example, TrackMate employs this framework so that segmentation, linking, and tracking steps are configurable via plugins.

The future of the ImageJ ecosystem

A number of pressing issues have been addressed with the advent of ImageJ2. On top of the list is the key question, “Will legacy macros and plugins still work?” The answer is yes since ImageJ2 must support the most important part of the ImageJ ecosystem: its thriving community, which includes the vast number of legacy macros and plugins it made. We foresee a steady shift to ImageJ2 as the new architecture not only makes writing plugins easier than before, but also allows the same plugins to be accessed from other software applications. We also foresee more and more software applications adding a layer of interoperability with ImageJ, particularly its ImageJ OPS library (http://imagej.net/OPS), which provides unified interfaces for basic image manipulations – a *lingua franca* for implementations of image-processing algorithms. Such interoperability has been made substantially easier by the SciJava project, with its generic, easily adaptable infrastructure for supporting plugins, among many other convenience functions. We expect the SciJava project to play a more fundamental role in scientific software development beyond ImageJ; it is already in use by core ImageJ2 components, including SCIFIO and ImageJ OPS, to provide powerful extension points. We predict that the same mechanism could, for example, be used to extend the Trainable Segmentation plugin to support user-provided features. Driven by the demand for concrete scientific software development, we expect SciJava to be developed and applied in useful and surprising ways to provide the functionality required in a wide range of software projects.

Although many scientific image-processing software projects are already able to interact with ImageJ in one form or another, interoperability is still an ongoing struggle. ImageJ has only recently begun employing the power of advanced image processing libraries such as ITK (Benmansour et al. 2012, Yoo et al. 2002), VTK (Sacha et al. 2003, http://ij-plugins.sf.net/), VIGRA (Köthe 2000), and OpenCV (Bradski 2000), but further work towards interoperability is still needed while the reciprocal relationship –calling ImageJ from said libraries– still has to be developed. And, there are many more active scientific software projects that cannot interoperate yet. Rather than “reinventing the wheel,” the emphasis in the next years of scientific software development should be to combine efforts and make it possible to benefit from one another, for the common goal of supporting scientific research.

Acknowledgments

The authors would like to thank Stephan Saalfeld for the CLAHE example image; Ignacio Arganda-Carreras for the trainable segmentation example image; Stephan Preibisch for the stitching sample image; and Christian Dietz for his work on the ImageJ KNIME node. We also acknowledge the input of members of LOCI on this manuscript, particularly Aparna Pal. Finally, the authors gratefully acknowledge the entire community of ImageJ software developers and users, whose efforts have made the ecosystem flourish.

Acknowledgments

Grant support:

grant sponsor: National Institutes of Health, grant number: RC2 GM092519

grant sponsor: Wellcome Trust, grant number: 095931

grant sponsor: National Science Foundation, grant number: 1148362

Abbreviations

API	Application Programming Interface
NIH	National Institute of Health

References

Allan C, Burel J, Moore J, Blackburn C, Linkert M, Loynton S, MacDonald D, Moore WJ, Neves C, Patterson A, Porter M, Tarkowska A, Loranger B, Avondo J, Lagerstedt I, Lianas L, Leo S, Hands K, Hay RT, Patwardhan A, Best C, Kleywegt GJ, Zanetti G, Swedlow JR. OMERO: flexible, model-driven data management for experimental biology. Nature Methods. 2012; 9(3):245–253. [PubMed: 22373911]

Arganda-Carreras I. bUnwarpj: Consistent and elastic registration in imagej, methods and applications. Second ImageJ User & Developer Conference. 2008

Austenfeld M, Beyschlag W. A Graphical User Interface for R in a Rich Client Platform for Ecological Modeling. Journal of Statistical Software. 2012; 49(4):1–19.

Benmansour, F., Longair, M., Turetken, E., Schindelin, J., Fua, P. Bioimage Analysis Workshop, 2012. Barcelona, Spain: 2012. ITK Wrapper for Fiji..

Berthold MR, Cebron N, Dill F, Gabriel TR, Kötter T, Meinl T, Ohl P, Sieb C, Thiel K, Wiswedel B. KNIME: The Konstanz information miner. Data Analysis, Machine Learning and Applications. 2008:319–326.

Bradski G. The OpenCV library. Doctor Dobbs Journal. 2000; 25(11):120–126.

Burger, W., Burge, MJ. Principles of Digital Image Processing. Springer; New York: 2010. p. 1-221.

Cardona A, Saalfeld S, Schindelin J, Arganda-Carreras I, Preibisch S, Longair M, Tomancak P, Hartenstein V, Douglas RJ. TrakEM2 software for neural circuit reconstruction. PLoS One. 2012; 7(6):e38011. [PubMed: 22723842]

Carpenter AE, Jones TR, Lamprecht MR, Clarke C, Kang IH, Friman O, Guertin DA, Chang JH, Lindquist RA, Moffat J, Golland P, Sabatini DM. CellProfiler: image analysis software for identifying and quantifying cell phenotypes. Genome Biology. 2006; 7:R100. [PubMed: 17076895]

Castleman, K. Digital Image Processing. Prentice Hall Press; Upper Saddle River: 1996. p. 1-667.

de Chaumont F, Dallongeville S, Chenouard N, Hervé N, Pop S, Provoost T, Meas-Yedid V, Pankajakshan P, Lecomte T, Montagner YL, Lagache T, Dufour A, Olivo-Marin J. Icy: an open bioimage informatics platform for extended reproducible research. Nature Methods. 2012; 9(7): 690–696. [PubMed: 22743774]

Collins TJ. ImageJ for microscopy. BioTechniques. 2007; 43(1 Suppl):25–30.

Cristy, J., Bergougnoux, K., Bogart, R., Peterson, JW., Brown, N., Chiarappa, M., Crimmins, TR., Zimmermann, A. ImageMagick. 1994. Currently available at http://ftp.x.org/contrib/applications/ImageMagick/ImageMagick-3.0.tar.gz

Delgado-Gonzalo R, Thevenaz P, Seelamantula CS, Unser M. Snakes with an ellipse-reproducing property. Image Processing, IEEE Transactions on. 2012; 21(3):1258–1271.

Eliceiri KW, Berthold MR, Goldberg IG, Ibáñez L, Manjunath BS, Martone ME, Murphy RF, Peng H, Plant AL, Roysam B, Stuurman N, Swedlow J, Tomancak P, Carpenter AE. Biological imaging software tools. Nature methods. 2012; 9(7):697–710. [PubMed: 22743775]

Forster B, Van De Ville D, Berent J, Sage D, Unser M. Complex wavelets for extended depth-of-field: A new method for the fusion of multichannel microscopy images. Microscopy Research and technique. 2004; 65(1-2):33–42. [PubMed: 15570586]

Hegge S, Kudryashev M, Smith A, Frischknecht F. Automated classification of Plasmodium sporozoite movement patterns reveals a shift towards productive motility during salivary gland infection. Biotechnology Journal. 2009; 4:903–913. [PubMed: 19455538]

Henriksson J, Hench J, Tong YG, Johansson A, Johansson D, Bürglin TR. Endrov: an integrated platform for image analysis. Nature methods. 2013; 10(6):454–456. [PubMed: 23722203]

Hibbard B. VisAD: Connecting people to computations and people to people. ACM SIGGRAPH Computer Graphics. 1998; 32(3):10–12.

Inoue S. Video image processing greatly enhances contrast, quality, and speed in polarization-based microscopy. The Journal of Cell Biology. 1981; 89(2):346–356. [PubMed: 6788777]

Kaynig V, Fuchs T, Buhmann JM. Neuron geometry extraction by perceptual grouping in sstem images. Computer Vision and Pattern Recognition (CVPR), June 2010 IEEE Conference on. 2010:2902–2909.

Konstantinides K, Rasure JR. The Khoros software development environment for image and signal processing. Image Processing, IEEE Transactions on. 1994; 3(3):243–252.

Köthe U. Generische Programmierung für die Bildverarbeitung. Dissertation, Fachbereich Informatik, Universität Hamburg. 2000 ISBN: 3-8311-0239-2.

Linkert M, Rueden CT, Allan C, Burel JM, Moore W, Patterson A, Loranger B, Moore J, Neves C, MacDonald D, Tarkowska A, Sticco C, Hill E, Rossner M, Eliceiri K, Swedlow JR. Metadata matters: access to image data in the real world. The Journal of cell biology. 2010; 189(5):777–782. [PubMed: 20513764]

Long F, Zhou J, Peng H. Visualization and analysis of 3D microscopic images. PLoS Computational Biology. 2012; 8(6):e1002519. [PubMed: 22719236]

Longair MH, Baker DA, Armstrong JD. Simple Neurite Tracer: Open Source software for reconstruction, visualization and analysis of neuronal processes. Bioinformatics. 2011; 27(17): 2453–2454. [PubMed: 21727141]

Meijering E, Smal I, Danuser G. Tracking in molecular bioimaging. Signal Processing Magazine IEEE. 2006; 23(3):46–53.

Möller, B., Greß, O., Posch, S. Computer Vision Systems. Springer; Berlin Heidelberg: 2011. Knowing what happened-automatic documentation of image analysis processes.; p. 1-10.

Pham DL, Xu C, Prince JL. Current methods in medical image segmentation. Annual review of biomedical engineering. 2000; 2(1):315–337.

Pietzsch T, Preibisch S, Tomančák P, Saalfeld S. ImgLib2—generic image processing in Java. Bioinformatics. 2012; 28(22):3009–3011. [PubMed: 22962343]

Pitrone P, Schindelin J, Stuyvenberg L, Preibisch S, Weber M, Eliceiri KW, Huisken J, Tomancak P. OpenSPIM-an open access platform for light sheet microscopy. Nature Methods. 2013; 10:598–599. [PubMed: 23749304]

Preibisch S, Saalfeld S, Tomancak P. Globally optimal stitching of tiled 3D microscopic image acquisitions. Bioinformatics. 2009; 25(11):1463–1465. [PubMed: 19346324]

Sacha JP, Cockman MD, Dufresne TE, Trokhan D. Quantification of regional fat volume in rat MRI. Medical Imaging. 2003; 2003:289–297.

Sage, D., Prodanov, D., Tinevez, JY., Schindelin, J. MIJ: Making Interoperability Between ImageJ and Matlab Possible.. ImageJ User & Developer Conference; Luxembourg. 24-26 October 2012; 2012.

Schindelin J, Arganda-Carreras I, Frise E, Kaynig V, Longair M, Pietzsch T, Preibisch S, Rueden C, Saalfeld S, Schmid B, Tinevez J, White DJ, Hartenstein V, Eliceiri K, Tomancak P, Cardona A. Fiji: an open-source platform for biological-image analysis. Nature Methods. 2012; 9(7):676–682. [PubMed: 22743772]

Schmid B, Schindelin J, Cardona A, Longair M, Heisenberg M. A high-level 3D visualization API for Java and ImageJ. BMC bioinformatics. 2010; 11(1):274. [PubMed: 20492697]

Schneider CA, Rasband WS, Eliceiri KW. NIH Image to ImageJ: 25 years of image analysis. Nature Methods. 2012; 9(7):671–675. [PubMed: 22930834]

Sluder, G., Wolf, D. Methods in Cell Biology. Vol. 81. Academic Press; 2007. Digital Microscopy; p. 1-632.

Stuurman N, Amdodaj N, Vale R. Micro-Manager: Open Source software for light microscope imaging. Microscopy Today. 2007; 15(3):42–43.

Yoo TS, Ackerman MJ, Lorensen WE, Schroeder W, Chalana V, Aylward S, Metaxas D, Whitaker R. Engineering and algorithm design for an image processing api: a technical report on itk-the insight toolkit. Studies in health technology and informatics. 2002:586–592. [PubMed: 15458157]

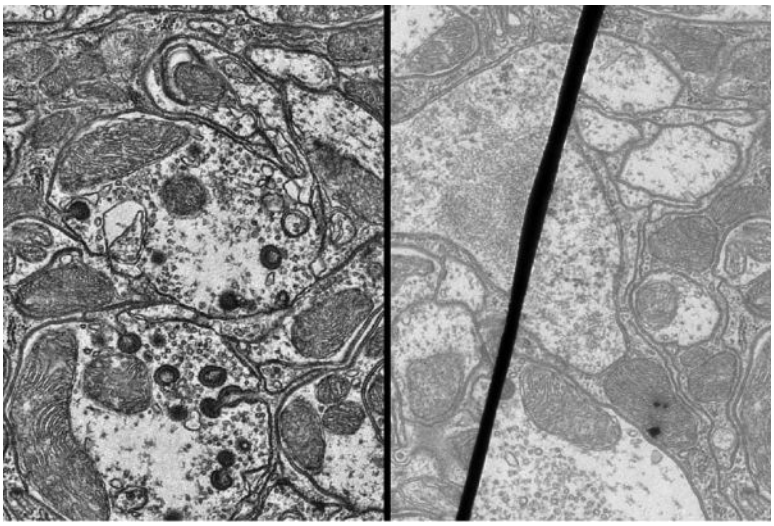

Figure 1.

An unprocessed electron micrograph (left) has minimal contrast, making feature detection difficult. After running the Contrast-Limited Adaptive Histogram Equalization (CLAHE) plugin in ImageJ, the resulting image (right) is suitable for further analysis. The CLAHE plugin has three parameters: block size determines the local-region extents used for histogram equalization; bins determines the number of histogram bins to use in equalization; and max slope limits the maximum changes in contrast in the intensity transfer function. For this image, the following parameters were used: block, 50; bins, 256; max slope, 2.5.

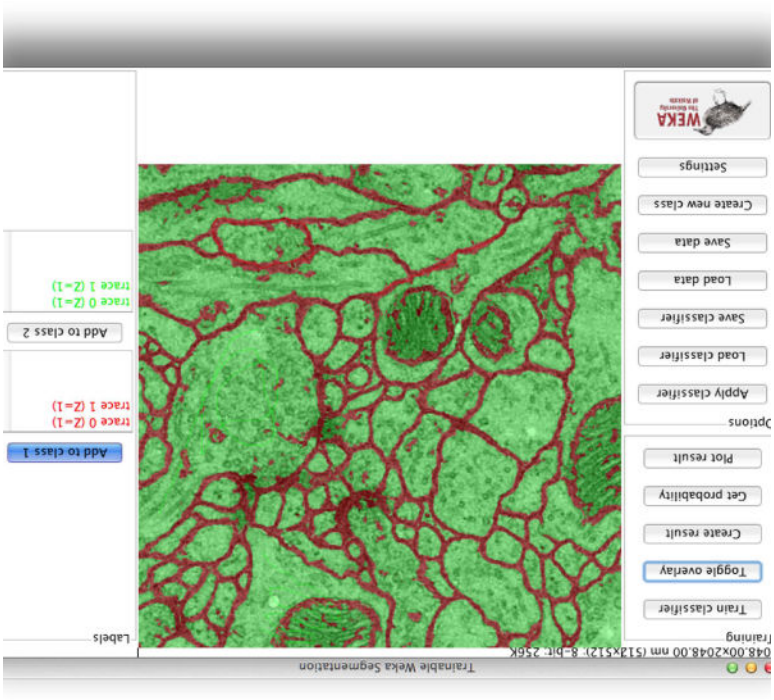

Figure 2.

An electron micrograph segmented with the Trainable Weka Segmentation plugin. Solid red lines (class 1: trace 0, 1) and green lines (class 2: trace 0, trace 1) were added manually by the user. The image regions covered by these traces make up the training sets passed to a WEKA classification algorithm. After training, the classifier can be applied to any input dataset. Applying the classifier to an image results in colored regions corresponding to the trained classes, as pictured here.

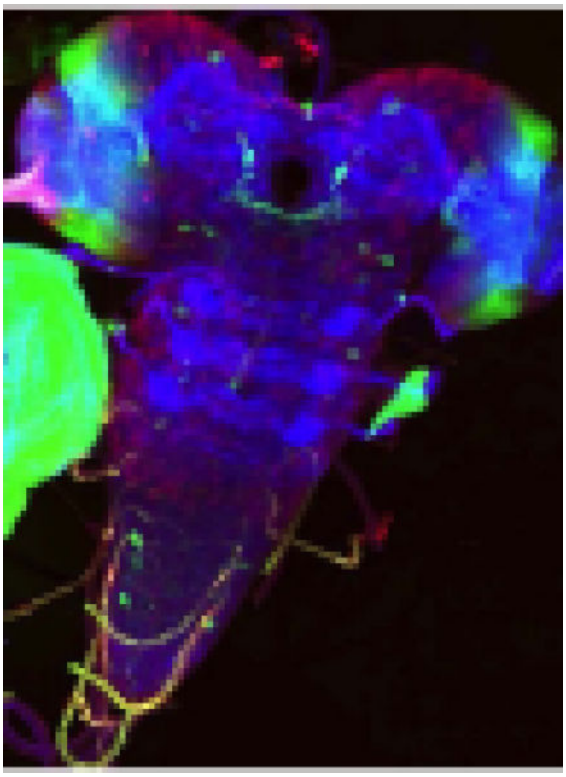

Figure 3.
Drosophila larval nervous system, stitched from a 2 × 3 grid of images assembled in ImageJ (Preibisch 2009). This plugin reads tile coordinates from metadata, then computes the overlap between adjacent tiles to determine each image's output coordinates. Overlapping regions are blended to create a uniform result, allowing visualization of the complete organism at a resolution greater than would otherwise be possible.

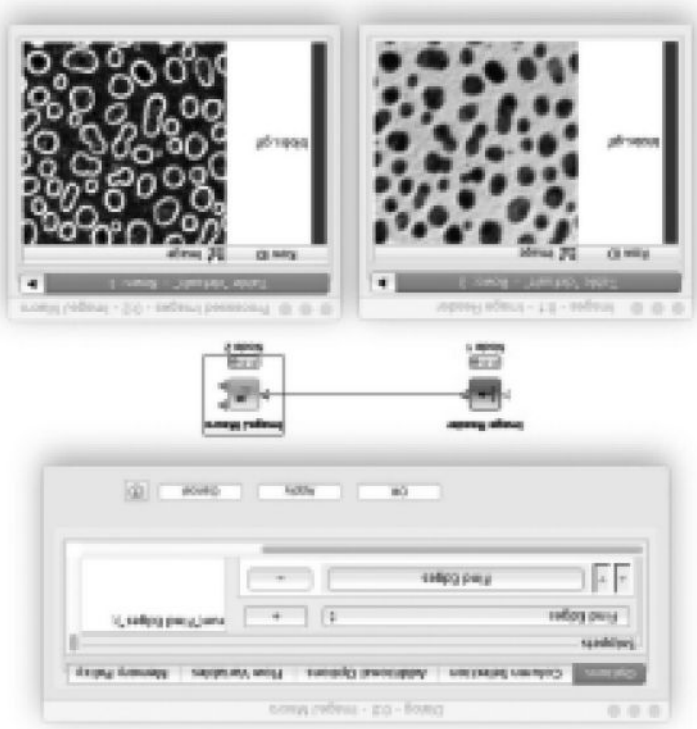

Figure 4.

ImageJ macro executed in a KNIME workflow. The Find Edges execution shown here, used to identify regions of high contrast, is one of several prepared functions bundled with the ImageJ KNIME node. This node is essentially an ImageJ script editor that is capable of running any ImageJ macro code that is headless-compatible (not requiring a user interface). In this way, users gain access to a significant number of ImageJ functions, coupled with the reproducibility and documentation inherent in KNIME workflows.